This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2760251, IEEE Access

A DAQM-based Load Balancing Scheme for High Performance Computing Platforms

Kaijun Yang, Meng Li, Guchuan Zhu, *Senior Member, IEEE,* and Yvon Savaria, *Fellow, IEEE*

***Abstract*—This paper addresses the load balancing problem, which is one of the key issues in high-performance computing (HPC) platforms. A novel method, called decentralized active queue management (DAQM), is proposed to provide a fair task distribution in a heterogeneous computing environment for HPC platforms. An implementation of the DAQM is presented, which consists of an ON-OFF queue control and a utility maximization-based coordination scheme. The stability of the queue control scheme and the convergence of the algorithm for utility maximization have been assessed by rigorous analysis. To demonstrate the performance of the developed queueing control system, numerical simulations are carried out and the obtained results confirm the efficiency and the viability of the developed scheme.**

***Index Terms*—decentralized active queue management (DAQM), load balancing, high performance computing.**

I. Introduction

Driven by the ever increasing complexity of applications and a stringent requirement in terms of Quality of Service (QoS), high performance computing (HPC) platforms have received considerable attention over the past decades [1], [2], [3]. Nowadays, HPC systems have been employed to support a great variety of scientific and commercial applications, such as data mining, computational biology, weather prediction, and mobile communication networks [4], [5]. Typically, an HPC platform is composed of multi-core computing units, e.g., GPUs, FPGAs and/or Network on Chips (NoCs), to offer a certain degree of parallelism and scalability [6]. Therefore, in order to fully utilize the capacity of HPC systems, an optimized load balancing is required to maximize computational efficiency.

In general, applications in an HPC platform can be divided into multiple tasks that can be executed on different nodes (GPU, FPGA, NoC, etc.). Hence, the goal of load balancing is to find a task mapping, which results in an approximately equal load distribution for each node [7]. The strategies for load balancing can be either static [8], [9] or dynamic [10], [11]. Generally, static load balancing (SLB) runs depending on previously obtained knowledge, while dynamic load balancing (DLB) continuously updates the information to make a decision. In scenarios where workloads are unpredictable, DLB is preferred to assure an adequate system performance, although SLB is simple to implement.

Much work has been dedicated to the development of algorithms for DLB. Different solutions have been proposed aiming at an efficient and fair resource allocation, while achieving adequate QoS levels [12], [13], [14]. From the control theoretical point of view, there basically exist two types of architectures for DLB: centralized closed-loop control and decentralized hierarchical control. The centralized closed-loop control requires complete and reliable knowledge of all application and platform parameters to make decisions. This architecture can provide good performance in grid computing, but it suffers from poor flexibility and scalability in addition to difficulties to handle unpredictable workloads. Another solution is the decentralized hierarchical control, in which each node runs locally in a closed-loop manner and contributes to the global performance. However challenges arise from decentralized control regarding, particularly, task distribution and load balancing among the processing nodes as it is in general hard for such a system to achieve stability and high utilization without global information.

In this paper, we adopt a decentralized architecture and tackle the challenge of balancing load among processing nodes. Inspired by the well-studied paradigm of active queue management (AQM), we introduce the concept of decentralized active queue management (DAQM). A DQAM-based scheme consists of two layers. At the lower layer, a local feedback control loop is introduced to leverage the basic feature of AQM to assure a stable operation at each processing node. At the upper layer, a coordination controller is applied to achieve a fair load balancing while maximizing resource utilization. The theoretical analysis shows that this method can provide a fair task distribution in a heterogeneous computing environment for HPC platforms.

The main contributions of this paper are:

- a DAQM-based load balancing strategy, which considers real-time computing capability of each processing node;
- an implementation of the DAQM, which can achieve a fair load balancing among processing nodes by minimizing the overall processing delay while maximizing resource utilization.

The remaining of the paper is organized as follows. Section II makes a brief review of some related work on task scheduling for HPC platforms and applications of AQM. Section III presents a queue management scheme for HPC platforms, namely DAQM. Section IV illustrates an implementation of DAQM mechanism. Then, in Section VI, the developed control mechanism is evaluated by simulation. Finally, some concluding remarks are provided in Section VII.

The authors are with the Department of Electrical Engineering, Polytechnique Montréal, P.O. Box 6079, Station Centre-Ville, Montreal, QC, Canada H3C 3A7.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2760251, IEEE Access

II. Related Work

A. Task Scheduling on HPC Platforms

With the development of HPC technology, it becomes possible to support computationally intensive applications with an aggregation of computing nodes instead of supercomputers. In such a heterogeneous computing environment, effectively mapping the tasks to minimize operating costs while respecting QoS constraints represents a real challenge.

One issue for task scheduling in this environment is the unpredictability of workloads. In recent years, much attention has been put on workload characterization in order to have a better understanding of workload features in terms of arrival rate and duration [15]. Such analysis can be found in [16], [17]. In [15], task classification is performed and utilized in resource provisioning, which allows for important energy savings while significantly reducing task scheduling delay.

Other works focus on the minimization of the makespan, which is defined as the maximum completion time of all tasks [18], [19], [20], [21], [22]. It is known that task scheduling is NP-complete in general [23], [24]. Thus, in the literature, several heuristic scheduling algorithms, such as Opportunistic Load Balancing (OLB) [18], [19], [25], Min-Min [20], [22], and Max-Min [20], [22], have been proposed to achieve suboptimal solutions. The intuition of OLB is to keep all computing resources as busy as possible. This may lead to poor makespan as it does not consider the expected task execution times. The Min-Min algorithm is based on the minimum completion time. The strategy behind the Min-Min algorithm is to select the task with the minimum completion time from all unscheduled tasks at each step, with the hope of obtaining a smaller makespan. The Max-Min algorithm is similar to the Min-Min, while its purpose is to minimize the penalties caused by tasks with longer execution times. For practical applications, suitable heuristic algorithms should be chosen by considering performance, efficiency, complexity, etc.

In our work, instead of characterizing the heterogeneity of workloads to perform task allocation, a generic decentralized active queue model is applied to achieve load balancing. By monitoring and adjusting the queue lengths instead of the behaviours of computing nodes, the overall average processing delay can be minimized.

B. Active Queue Management

Queueing theory is a mathematical model-based analysis framework. In general, its ultimate objective is to model the behavior of queueing systems from which it can take appropriate actions [26]. Queueing theory is extensively applied in many industrial sectors, in particular in information and communications technology (ICT) industries for system dimensioning, performance assessment, traffic engineering, etc. [27], [28]. By developing proper models such as $M/M/1$ queue [29], [30], $M/D/1$ queue [31], $M/G/1$ queue [32], $M/E_k/1$ queue [33], queuing length and waiting time can be predicted. It has been observed that long latency and delay variation in queueing networks, e.g, packet networks, can be induced by excess buffering of packets. In order to address these issues, several queue management schemes have been proposed and investigated, among which we can find the active queue management (AQM) [34].

The main objective of AQM is to maintain the queues at an adequate level, which represents a compromise between queueing delay and resource utilization. Benefiting from AQM, system performance can be improved and QoS can be guaranteed in an average sense. AQM is extensively used in TCP networks for congestion control. In this context, the AQM controller notifies traffic sources of congestion in order to reduce their transmission rates to avoid congestion and to reduce both queueing delay and packet loss [35], [36]. For example, the Early Random Drop and Random Early Detection (RED) mechanism [34] is a variation of the Random Drop mechanism aimed at avoiding congestion by predicting when it will occur rather than reacting to it. Many significant modifications have been adopted in RED, such as Adaptive RED, Gentle RED, BLUE, Random Early Marking (REM), and Double Slope RED (DSRED), in order to improve its performance. Most of these studies mainly focus on when and how to drop the arriving packets and to reduce RED sensitivity to parameter settings. Explored solutions rely on static thresholds which can be restrictive when they operate with sources.

Note that although the DAQM introduced in this work is greatly inspired by the paradigm of AQM, it behaves differently. Detailed explanations on the basic properties of DAQM are provided in the next section.

III. Architecture of DAQM for HPC Platforms

A. Basic Queueing Models for AQM

A queueing model is composed of three basic components: arrival process, queue, and service process, as illustrated in Fig. 1. These three components interact with each other and the information changes of each part will affect the behaviour of the others. For the arrival process of tasks, we often make assumptions that the interarrival times are independent and have an identical distribution, such as Poisson distribution. Tasks can get into queues one by one or in batches, which will vary in different applications. The behaviour of assigned tasks depends mainly on the nature of the associated operations. Some tasks may be impatient and leave after a while, such as the calls in telephone systems. Some tasks may wait in a line to be processed by the server. The queues are the core of queueing systems, which can be divided into infinite queues and finite queues. When the tasks go through the queues, they should all comply with some queueing rules that determine how tasks are processed. A queueing rule determines which tasks can get processed and which tasks can be dropped by the queue. Queueing rules vary in the applied models, such as first-come-first-served [37], last-come-first-served [38], priority service order [39], random service order [40], etc. The emphasis of the present work is put on the control theory-based approaches [35],[41].

There are two basic types of queueing systems: queueing systems with exponential arrivals and queueing systems with non-exponential arrival distributions. In order to study the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2760251, IEEE Access

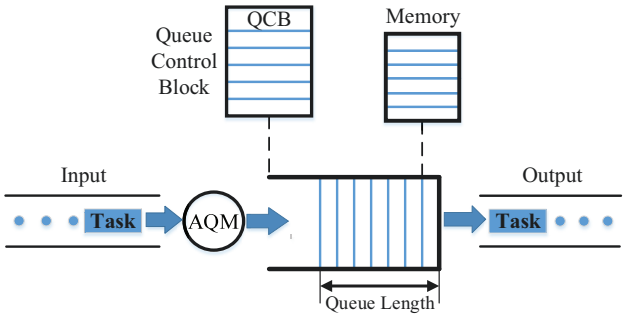


Fig. 1. Schematics of AQM for task flow management

transient performance of systems with queues of processing tasks, some approximate models have been developed, such as diffusion models [42], fluid flow models [43], and service time convolution. The flow conservation principle can be applied in single queue systems to describe the time-dependent queue dynamics. Let $x(t)$ be the average queue length and $\dot{x}(t) = dx(t)/dt$ denote the rate of change of the average queue length. Then, by conservation law, the rate of change of the average queue length in the system is equal to the difference between the average arrival and departure rates. The fluid dynamic queueing model is extensively used in network systems analysis and design (see, e.g., [42],[44]). It has been shown that this model matches very well many real-life situations [45], [46].

The approach of regulating queue lengths based on a fluid model is motivated by two main reasons: first, it is used to randomly compute periods of filling and emptying, and second, it relies on continuous incoming rates, unlike M/M/1 or M/G/1 queues based on discrete arrivals. As the traffic flow is composed of a huge number of particles, a sufficiently long discrete incoming traffic flow can be considered as a continuous process, which suits to the network architectures, as well as network traffic control. In the AQM model, it is common to assume that the incoming flow will be very large over a long time, which means that the queue may overflow. Thus, in order to make a network work smoothly, controllers in AQM are used to drop tasks waiting in queues, which may result in failure of running applications due to dependency. In our work, dropping tasks is not allowed and hence AQM is not suitable for our context.

In a queue model, both the time between successive arrivals and the service time are exponentially distributed. This model can be expressed by ordinary differential equations. The M/M/1 queue is one of the simplest models [30]. In an M/M/1 queue, the interarrivals are described by a Poisson process with mean $1/\mu_a$, the service times are exponentially distributed with a mean value $1/\mu_s$, and there is a single server. Tasks are served in the order of arrival. The queueing dynamics are given by

$$\dot{x}(t) = -\frac{x(t)}{1+x(t)}C(t) + \lambda(t), \tag{1}$$

where $\lambda(t)$ denotes the incoming traffic rate, and $C(t)$ is the service rate.

Remark III.1. *In the present work, the M/M/1 queue model is used in queueing control design and the validation of the proposed load balancing scheme. However, other queue models (for instance M/D/1,M/G/1) can also be applied and will yield similar results.*

B. Decentralized Active Queue Management

In the schemes based on AQM for network congestion control, the queues to be controlled are located at the bottleneck points, such as routers and switches. Moreover, it is commonly assumed that packet dropping is allowed and that the processing speed of the servers is known, or even can be controlled. However, this is not the situation for the considered workload balancing problems in HPC platforms. Specifically, this is because:

- The direction of information flows is reversed compared to the common network traffic control problems. In the considered computing platform, task flows enter into the master node and then the tasks are dispatched to multiple servers (or processing nodes). Therefore, queue control at the master node level becomes gradually impractical for larger and larger systems.
- Each sever can contain multiple processors (or cores) managed by local schedulers. Indeed, the processing rate of a node is time-varying and is hard to control.
- Task forwarding must be performed in a lossless manner, which means that discarding tasks is not allowed.

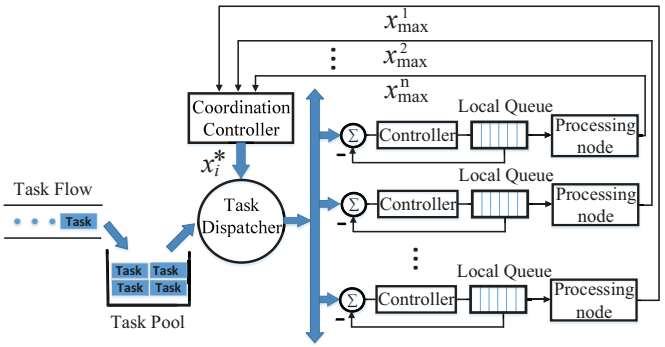


Fig. 2. Architecture of a System with Decentralized Active Queue Management

In order to develop suitable solutions for workload control in computing clusters, we introduce a novel concept, called decentralized active queue management (DAQM). Fig. 2 shows the schematic diagram of a DAQM system in the context of workload control in a computing cluster. In the paradigm of DAQM, each computing node manages a local queue. As in decentralized control schemes, there is a control component at the master node, which is used to maintain the queue length around a desired value. In order to simplify the implementation of task allocation strategy at the master node, a proportional distribution mechanism is applied. In this way, the task dispatcher at the master node assigns tasks proportionally according to the desired queue length of each computing node. The choice of the target queue length is critical because it

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2760251, IEEE Access

reflects a compromise between the processing delay and the efficiency of resource utilization.

IV. DAQM Implementation

In this section, we present an implementation of the DAQM for workload balancing in a computing cluster. The proposed scheme consists in a set of local queue controllers and a coordination controller at the cluster level for utility maximization. In the present work, a simple ON-OFF control scheme is used to achieve a stable queue length regulation under certain realistic assumptions, which coordinates the operation of the local queue controllers to achieve a global optimality, e.g., fair task distribution by minimizing the overall processing delay and maximizing the resource utilization. Let x^* be the desired reference of task distribution. The choice of x^* is crucial for the implementation of DAQM to guarantee that every local queue fairly receives tasks based on the current utility of its corresponding processing node. In the next section, we determine the value of x^* by the method of dual decomposition.

A. Utility Maximization-based Coordination Control: Computation of x^*

In the queue control scheme, the choice of the desired queue lengths has an important influence on systems performance. A small value of x^* indicates that a small processing delay is expected, while the utilization of the server may be low. Whereas, a high value of x^* leads to a high utilization efficiency, but the system may experience important delays. Moreover, in the environment of cluster computing, the workloads have to be fairly distributed over all the computing nodes to reduce the overall processing delay while maximizing the utilization of the cluster. To this end, the coordination control at the cluster level is formulated as a utility maximization problem. In this context, we consider the following utility function:

$$U(x) = \sum_{x_i \in \mathbb{R}^n} U_i(x_i), \tag{2}$$

where $x = \{x_1, x_2, ..., x_n\}$ is an allocation vector and x_i is the reference length of the ith queue. We choose a utility function of the following form:

$$U_i(x_i) = 1 - \left(1 - \frac{x_i}{x^i_{\max}}\right)^{w_i},\ 0 < x_i \leq x^i_{\max},\ 1 < \ w_i, \tag{3}$$

where $x^i_{\max}$ is the maximum queue size associated with processing node i. This utility function implies that when x_i becomes very close to $x^i_{\max}$, the value of its utility function tends to 1, corresponding to the highest utilization of the ith node. Let X_{total} be the maximum number of tasks on average in a task pool, then the considered utility maximization problem can be expressed as

$$\begin{aligned} &\text{maximize}_{\{x \in \mathbb{R}^n\}} && U(x) \\ &\text{subject to :} && x_i \in [0, x^i_{\max}], \forall i \in \{1, \ldots, n\}, \\ & && \sum_{i=1}^{n} x_i \leq X_{total}. \end{aligned} \tag{4}$$

To handle the coupling constraints in (4), we resort to the method of dual decomposition based on Lagrange relaxation [47]. Let us introduce the Lagrange function

$$\begin{aligned} L(x, \mu) &= U(x) + \mu\left(X_{total} - \sum_{i=1}^{n} x_i\right) \\ &= \sum_{i}^{n} L_i(x_i, \mu) + \mu X_{total}, \end{aligned} \tag{5}$$

where μ is the Lagrange multiplier and $L_i = U_i - \mu x_i$. The problem (4) can be reformulated as

$$\begin{aligned} &\text{maximize} && \sum_{i}^{n} L_i(x_i, \mu) + \mu X_{total} \\ &\text{subject to} && x_i \in [0, x^i_{\max}], \forall i = 1, ..., n. \end{aligned} \tag{6}$$

Then, the prime optimum of the original problem is given by

$$x^*_i(\mu) = \underset{x_i \in [0, x^i_{\max}]}{\arg\max}\ \left(L_i(x_i, \mu)\right), \tag{7}$$

which is unique due to the strict concavity of U_i. Moreover, considering the utility function U_i given in (3), the optimal solution for (7) is given by

$$x^*_i(\mu) = \max\left[0, x^i_{\max}\left(1 - \frac{\mu x^i_{\max}}{w_i}\right)^{\frac{1}{w_i - 1}}\right], \tag{8}$$

which is the explicit optimal value.

The master dual problem is given by

$$\begin{aligned} &\text{minimize} && g(\mu) = \sum_{i=1}^{n} g_i(\mu) + \mu X_{total} \\ &\text{subject to} && \mu \geq 0, \end{aligned} \tag{9}$$

where $g_i = L_i(x^*_i, \mu),\ i \in \{1, \ldots, n\}$, are the dual functions [47]. In order to obtain the primal solution, we apply the gradient method to (9), which gives

$$\mu(k+1) = \left[\mu(k) - \alpha_k\left(X_{total} - \sum_{i=1}^{n} x^*_i(\mu(k))\right)\right]^+, \tag{10}$$

where k is the iteration index, α_k is a sufficiently small positive step-size, and $[\cdot]^+$ denotes the projection onto the non-negative part.

B. Local Queue Control

In the present work, we consider first a local queueing control based on the fluid model given in (1). Furthermore, we impose the following assumption:

Assumption IV.1. *Assume that $\lambda(t) \in [0, \lambda_{\max}]$ and $C(t) \in (0, C_{\max}]$, where $\lambda_{\max}$ is the maximum allowed input task flow rate and $C_{\max}$ is the maximum service rate of the processing node. Furthermore, for every fixed $t_0 > 0$, it holds*

$$x^* < \int_{t_0}^{\infty} \lambda(\tau)\ d\tau < \int_{t_0}^{\infty} C(\tau)\ d\tau < \infty, \tag{11}$$

where x^* is the desired queue length.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2760251, IEEE Access

This assumption states essentially that the input flow is bounded, and the server has a limited capacity while it will never be completely idle. Specifically, the assumption $x^* < \int_{t_0}^{\infty} \lambda(\tau)\, d\tau$ is a persistent excitation requirement. Under this condition, the queue length $x(t)$ will reach the reference x^* in finite-time if $x(t_0) < x^*$ [46]. This implies that the server is sufficiently utilized in the long run. The assumption $\int_{t_0}^{\infty} \lambda(\tau)\, d\tau < \int_{t_0}^{\infty} C(\tau)\, d\tau$ is a lossless condition, which can guarantee the aim that no task will be discarded in a queue may be achievable. In addition, as $\lim_{x\to\infty} \frac{x}{x+1} C = C$, if the queue length become very large, the derivative of queue length will turn into a negative value, which implies that the queue length will decrease. Thus, this condition ensures also that the queue length will never grow to infinity.

Note that if $\int_{t_0}^{\infty} \lambda(\tau)\, d\tau < x^*$ and $x(t_0) < x^*$, then all the input tasks will pass through the queue. The queue length will stay around $\overline{x} = \frac{\overline{\lambda}}{\overline{C} - \overline{\lambda}}$, where $\overline{C}$ and $\overline{\lambda}$ are, respectively, the average server rate and the average input task flow rate. This means that the queueing control has no effect. Meanwhile, it indicates that the server might not be sufficiently utilized.

For the purpose of lossless queueing control, we developed a control scheme shown in Fig. 3. The following ON-OFF scheme is used in this controller:

$$u = \begin{cases} 1, & x - x^* \le 0, \\ 0, & x - x^* > 0. \end{cases} \tag{12}$$

Note that, to achieve a lossless queue length regulation, the tasks will be routed back to the input buffer when $u = 0$. When $u = 1$, the input buffer will be flushed out and all the tasks will be forwarded to the queue. The stability of this control scheme is assessed below.

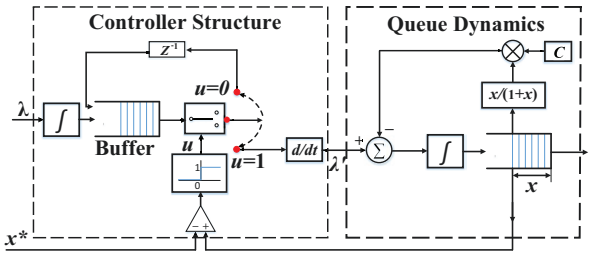


Fig. 3. Schematics of the ON-OFF queueing control system.

Theorem IV.1. *Consider a queueing system governed by* (1) *satisfying the conditions specified in Assumption IV.1. The queuing length of such a system with the control given in* (12) *converges to the reference queue length x^* as $t \to \infty$ or has x^* as its upper bound.*

Proof. When $x \ge x^*$, let $V = \frac{1}{2}(x - x^*)^2$ be a Lyapunov candidate function. Noting that in this case $u = 0$, the time-derivative of V along the trajectory of the controlled system is given by

$$\dot{V}(t) = \frac{x x^* - x^2}{1 + x} C \le 0.$$

By virtue of Barbalat's lemma [48], we can conclude that $|x - x^*| \to 0$ as $t \to \infty$.

When $x < x^*$, the controller $u = 1$. The controlled system becomes:

$$\dot{x} = -\frac{x}{1 + x} C + \lambda. \tag{13}$$

We apply the Comparison principle [48] to estimate x. Let us first construct a new system

$$\dot{y} = -C + \lambda, \quad y(t_0) = x(t_0), \tag{14}$$

where t_0 represents the initial time and $y(t_0)$ and $x(t_0)$ are the initial values of y and x, respectively. Let $v = x - y$. According to (13) and (14), we can obtain

$$\dot{v} = C - \frac{x}{1 + x} C \ge 0, \quad v(t_0) = 0. \tag{15}$$

Therefore, $x \ge x(t_0) + \int_{t_0}^{t} \lambda(\tau)\, d\tau - \int_{t_0}^{t} C(\tau)\, d\tau$. Based on the condition (11), there exists T such that $x(T) = x^*$, otherwise $x(t) < x^*$, for $t > t_0$. In the latter situation, based on the first part of the proof, we have $|x - x^*| \to 0$ as $t \to \infty$. □

V. Convergence Analysis and Algorithm for Utility Maximization

In section (IV), an optimal solution to the problem (4), x_i^*, has been obtained. Specifically, x_i^* can be computed through an iterative process that updates the Lagrange multiplier μ via (10). Furthermore, μ is obtained by solving the master dual problem (9). In this section, we first prove that the optimal solution of (9) can be approximated with a solution in finite iterations by applying the analysis technique used in [47], [49]. Then, two types of step size rules are presented so as to obtain x_i^* efficiently and explicitly within finite recursion steps.

Define

$$g_{best}^{(k)} = \min\{g(\mu(1)), ..., g(\mu(k))\}, \tag{16}$$

where $g_{best}^{(k)}$ is the best objective value found in k iterations. Since $g_{best}^{(k)}$ is decreasing and positive, it admits a limit. Suppose that μ^* is an optimal point to the problem (9). The desired objective is to approximate the optimal value $g(\mu^*)$ by $g_{best}^{(k)}$, i.e, $\lim_{k\mapsto\infty} g_{best}^{(k)} - g(\mu^*) \to 0$. We begin with estimating the distance between $\mu(k+1)$ and μ^*,

$$\begin{aligned}
&|\mu(k+1) - \mu^*|^2 \\
=&\left| \left[\mu(k) - \alpha_k (X_{total} - \sum_{i=1}^{n} x_i^*(\mu(k))) \right]^+ - \mu^* \right|^2 \\
=&|\mu(k) - \mu^*|^2 - 2\alpha_k \left(X_{total} - \sum_{i=1}^{n} x_i^*(\mu(k)) \right) (\mu(k) - \mu^*) \\
&+ \alpha_k^2 \left(X_{total} - \sum_{i=1}^{n} x_i^*(\mu(k)) \right)^2 \\
\le&|\mu(k) - \mu^*|^2 - 2\alpha_k \left(g(\mu(k)) - g(\mu^*) \right) \\
&+ \alpha_k^2 \left(X_{total} - \sum_{i=1}^{n} x_i^*(\mu(k)) \right)^2 .
\end{aligned}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2760251, IEEE Access

Applying the above inequality recursively, we obtain

$$|\mu(k+1)-\mu^*|^2 \le |\mu(1)-\mu^*|^2 - 2\sum_{i=1}^{k}\alpha_i\left(g(\mu(i))-g(\mu^*)\right) + \sum_{j=1}^{k}\alpha_j^2\left(X_{total}-\sum_{i=1}^{n}x_i^*(\mu(j))\right)^2.$$

As $|\mu(k+1)-\mu^*|^2 \ge 0$, it yields

$$2\sum_{i=1}^{k}\alpha_i\left(g(\mu(i))-g(\mu^*)\right) \le |\mu(1)-\mu^*|^2 + \sum_{j=1}^{k}\alpha_j^2\left(X_{total}-\sum_{i=1}^{n}x_i^*(\mu(j))\right)^2. \quad (17)$$

By the inequality (17), we can obtain the estimate of $g_{best}^{(k)} - g(\mu^*)$:

$$g_{best}^{(k)}-g(\mu^*) \le \frac{|\mu(1)-\mu^*|^2+\sum_{j=1}^{k}\alpha_j^2(X_{total}-\sum_{i=1}^{n}x_i^*(\mu(j)))^2}{2\sum_{i=1}^{k}\alpha_i} \le \frac{|\mu(1)-\mu^*|^2+\sum_{j=1}^{k}\alpha_j^2(X_{total}+\sum_{i=1}^{n}x_{\max}^i)^2}{2\sum_{i=1}^{k}\alpha_i}.$$

In order to verify that $g_{best}^{(k)}$ can approximate the optimal value $g(\mu^*)$, two step size rules are considered.

Constant step size. If $\alpha_k = h$ is a constant, independent of k, then we obtain

$$g_{best}^{(k)} - g(\mu^*) \le \frac{|\mu(1)-\mu^*|^2 + h^2k(X_{total}+\sum_{i=1}^{n}x_{\max}^i)^2}{2hk}.$$

For a sufficiently large k, $g_{best}^{(k)} - g(\mu^*) \le h(X_{total} + \sum_{i=1}^{n}x_{\max}^i)^2$. We can choose a very small h to approximate the optimal value $g(\mu^*)$.

Square summable but not summable. If the step size satisfies

$$\sum_{k=1}^{\infty}\alpha_k^2 < \infty, \quad \sum_{k=1}^{\infty}\alpha_k = \infty. \quad (18)$$

we can obtain the following estimate:

$$g_{best}^{(k)}-g(\mu^*) \le \frac{|\mu(1)-\mu^*|^2+\sum_{i=1}^{k}\alpha_i^2(X_{total}+\sum_{i=1}^{n}x_{\max}^i)^2}{2\sum_{i=1}^{k}\alpha_i}.$$

Hence, when k tends to infinity, $g_{best}^{(k)} - g(\mu^*)$ converges to 0.

Based on the above analysis of the two cases, $g_{best}^{(k)}$ can converge to $g(\mu^*)$ when k tends to ∞. Thus, we can approximate the optimal problem (4) based on $g_{best}^{(k)}$. An implementation of the considered utility maximization problem is given by Algorithm 1.

Algorithm 1 Utility Maximization

Require: queue capacities ($x_{\max}^i$ for all $i = 1, ..., n$), α_k a sequence of steps;
Ensure: $0 \le x_i \le x_{\max}^i$, $\mu > 0$, $\mu_{best} = \mu$;

```
Set k = 0, and μ(1) equals to some nonnegative value
for all k = 1 : n do
    for all i = 1 : n do
        compute x_i^*(μ) ∈ arg max_{x_i ∈ [0, x_max^i]} (L_i(x_i, μ))
    end for
    μ ∈ arg min Σ_{i=1}^n g_i(μ) + μ X_total
    μ ← [μ − α_k (X_total − Σ_{i=1}^n x_i^*(μ))]^+
end for
```

VI. Performance Evaluation

In this section, the performance of the queueing control system developed in Section IV is evaluated through numerical simulations with Matlab. In Section VI-A, a single queueing control is validated by using a randomly distributed input and a sine wave input. In Section VI-B, the DAQM control is evaluated with a varying input.

A. Simulation of Single Queueing Control

Consider a queue with a processing capacity of $C = 20$, and suppose that the input task flow is randomly distributed as shown in Fig. 4(a). Note that in the simulation, all the variables and parameters are in normalized coordinates. It can be seen from Fig. 4(b) that an uncontrolled queue tends to overflow very quickly. Therefore, adequate controls are required to make the system stable. In this simulation, the reference queue lengths of the two servers are set to $x^* = 50$ and $x^* = 30$, respectively. As shown in Fig. 4(c), by using the proposed ON-OFF control mechanism, the queue length can be stabilized around the desired level or be kept below the reference queue length in both cases.

Furthermore, a sine wave input flow rate is used to test the controlled queue as shown in Fig. 5(a). The simulation result given in Fig. 5(c) for a setup with $C = 40$ shows that the ON-OFF control mechanism works well. For both references, $x^* = 20$ and $x^* = 40$, the queue length can be stabilized around the desired level.

B. Validation of the DAQM Control Scheme

As presented in the previous sections, the maximum queue size of a processing node corresponds to the processing capacity. Given an admissible delay, $x_{\max}^i$ can be fixed according to its processing capacity, e.g., the processing speed and the bus width (32-bit or 64-bit). Considering a HPC platform consisting of multiple processing nodes, a design preference can be introduced to allocate the resources and maximize the system utilization. In general, nodes with higher processing rates can be assigned larger queue length references to achieve a better resource utilization.

Suppose that there are three computing nodes in one cluster. We then consider a scenario in which the total input varies with time, leading to the reconfiguration of the reference queue

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2760251, IEEE Access

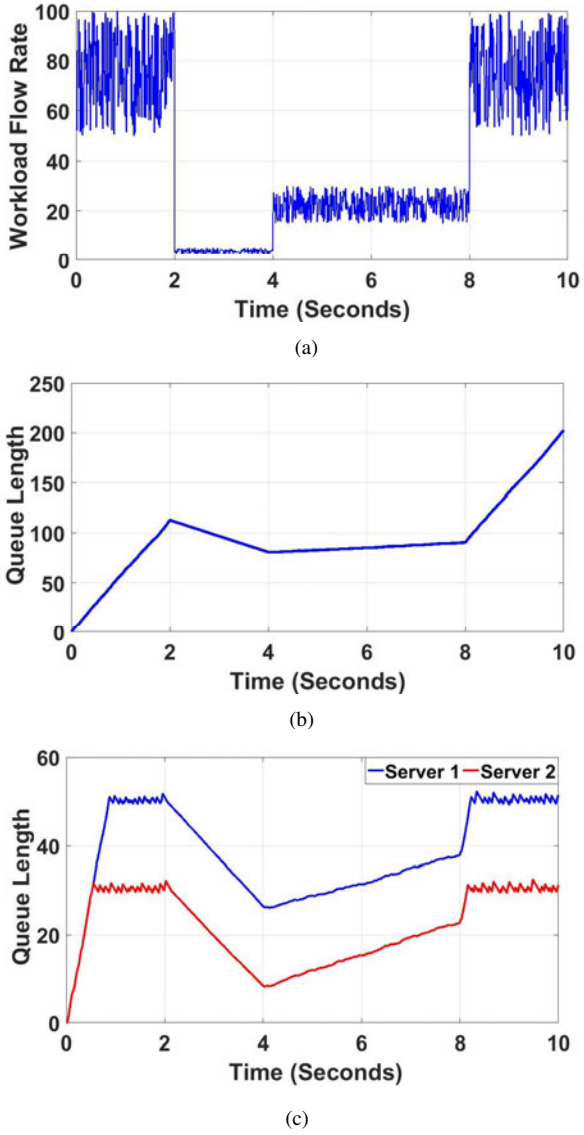


Fig. 4. Uniform random workload: (a) input flow rate; (b) uncontrolled queue; (c) Controlled queue: $C = 20$; Server 1: $x^* = 50$; Server 2: $x^* = 30$.

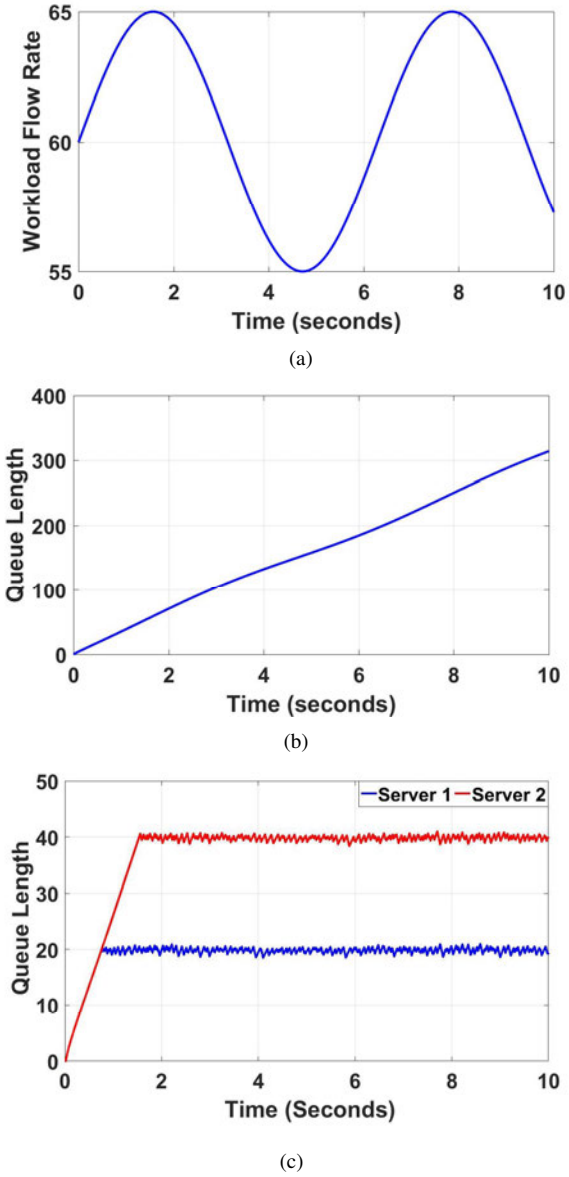


Fig. 5. Sine wave workload: (a) input flow rate; (b) uncontrolled queue; (c) controlled queue: $C = 40$; Server 1: $x^* = 20$; Server 2: $x^* = 40$.

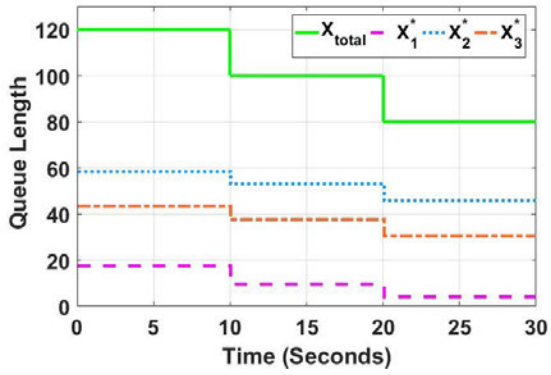
length of each node. Fig. 6 shows a simulation result, in which the reference queue lengths of three nodes are reported. In this simulation, it is assumed that the processing capacity of each node is constant. In addition, the node with a larger maximum queue size is assigned with more tasks to maximize resource utilization.

For a given configuration, the reference queue length for each processing node can be obtained through iterations as shown in Fig. 7(a). Finally, the iterative solution will converge and simultaneously the Lagrange multiplier tends to zero as shown in Fig. 7(b). In this case, after 2000 iterations, the reference queue length for each processing node can be obtained.

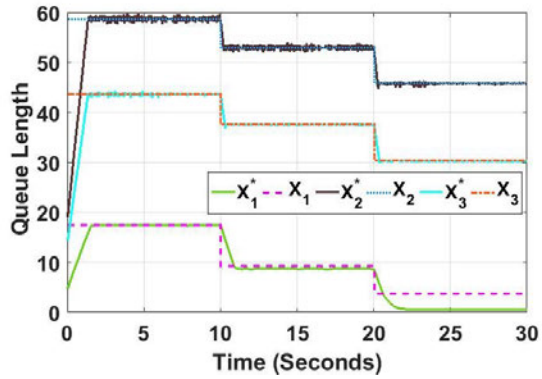
It can be seen from Fig. 6(a) that resource allocation is reconfigured according to the updated average input in total. Accordingly, the flow dispatched to each node is adjusted. As shown in Fig. 6(b), the queue length of each node is stabilized around or blew the reference queue length. Finally, the input workload and the input flow of each node are given in Fig. 8. The simulation results show that the workloads are fairly distributed over the three nodes and the queue lengths are all kept at an adequate level.

It can be observed from the above simulations that the queue length can be stabilized around the desired level when the task input or the desired reference varies. The control scheme can quickly adapt the change of configurations. Since the processing capacity does not change significantly in a short time, it is assumed to be constant in the simulation. In this case, any significant processing capacity change can be treated as a system reconfiguration. Therefore, the simulation results still hold. Furthermore, by taking advantage of the decentralized architecture, the system can be scaled easily by adding or removing processing nodes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2760251, IEEE Access

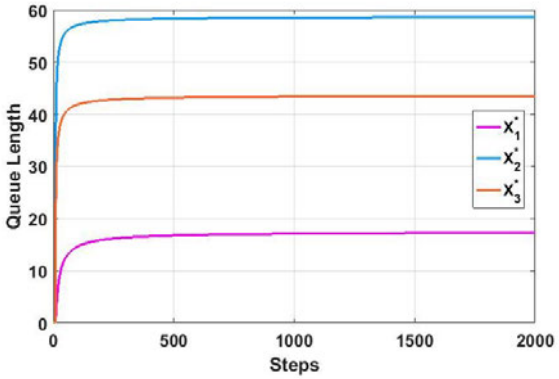


(a)

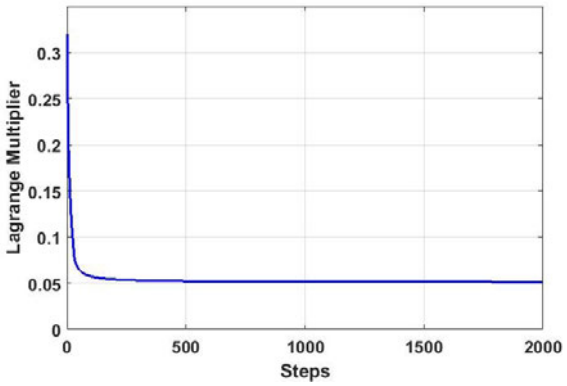


(b)

Fig. 6. Queuing control with variable reference queue length: (a) X_{total} subject to $x^1_{\max} = 20$, $x^2_{\max} = 60$, $x^3_{\max} = 45$; (b) controlled queue lengths.



(a)



(b)

Fig. 7. Computation results of the reference queue length: (a) reference queue length with the configuration of $x^1_{\max} = 30$, $x^2_{\max} = 60$, $x^3_{\max} = 50$, and $X_{total} = 100$; (b) convergence of the Lagrange multiplier.

VII. Conclusion

This work presented a DAQM-based control scheme, consisting in local queueing control and coordination control for workload balancing in the context of HPC platforms. An implementation of a DAQM-based control system was developed. It was shown that it can achieve a fair load balancing by controlling the queue lengths and dynamically adjusting the reference queue length for each node. Finally, the performance of the control scheme has been evaluated by numerical simulations, and the obtained results confirm the validity and feasibility of the proposed strategy.

In the present work, the same queueing-based control mechanism is used in all processing nodes. However, in a more generic setting, each processing node may have its own control model [50]. In our future work, the coordination of different control models will be conducted.

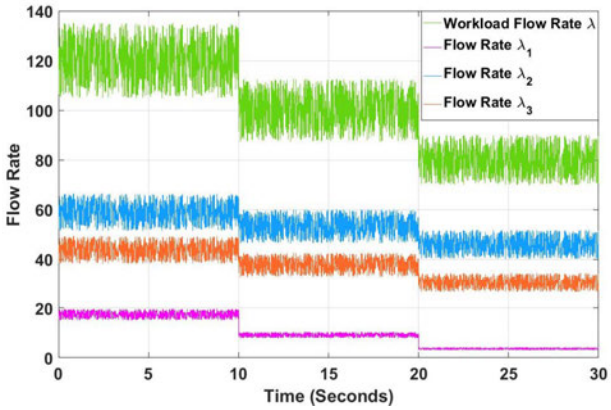


Fig. 8. Workload flow input and the flow dispatched to each processing node.

Acknowledgments

This work is financially supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) and in part by Huawei Technologies Canada Co., Ltd.

References

[1] A. J. Wallcraft, H. E. Hurlburt, E. J. Metzger, R. C. Rhodes, J. F. Shriver, and O. M. Smedstad, "Real-time ocean modeling systems," *Computing in Science Engineering*, vol. 4, no. 2, pp. 50–57, Mar 2002.

[2] A. Chinea, S. Grivet-Talocia, S. B. Olivadese, and L. Gobbato, "High-performance passive macromodeling algorithms for parallel computing platforms," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 3, no. 7, pp. 1188–1203, July 2013.

[3] X. Meng and V. Chaudhary, "A high-performance heterogeneous computing platform for biological sequence analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 9, pp. 1267–1280, Sept 2010.

[4] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 2867–2876, Nov 2014.

[5] F. D. Igual, M. Ali, A. Friedmann, E. Stotzer, T. Wentz, and R. A. van de

Geijn, "Unleashing the high-performance and low-power of multi-core DSPs for general-purpose HPC," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012, pp. 1–11.

[6] F. A. Escobar, X. Chang, and C. Valderrama, "Suitability analysis of FPGAs for heterogeneous platforms in HPC," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 600–612, Feb 2016.

[7] J. Watts and S. Taylor, "A practical approach to dynamic load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 3, pp. 235–248, Mar 1998.

[8] Y. Alexeev, A. Mahajan, S. Leyffer, G. Fletcher, and D. G. Fedorov, "Heuristic static load-balancing algorithm applied to the fragment molecular orbital method," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012, pp. 1–13.

[9] T. K. Ghosh, R. Goswami, S. Bera, and S. Barman, "Load balanced static grid scheduling using Max-Min heuristic," in *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on*, Dec 2012, pp. 419–423.

[10] V. Cardellini, M. Colajanni, and P. S. Yu, "Dynamic load balancing on web-server systems," *IEEE Internet Computing*, vol. 3, no. 3, pp. 28–39, May 1999.

[11] K. Son, S. Chong, and G. D. Veciana, "Dynamic association for load balancing and interference avoidance in multi-cell networks," *IEEE Transactions on Wireless Communications*, vol. 8, no. 7, pp. 3566–3576, July 2009.

[12] A. Y. Zomaya and Y.-H. Teh, "Observations on using genetic algorithms for dynamic load-balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 9, pp. 899–911, Sep 2001.

[13] Z. Zeng and B. Veeravalli, "Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks," *IEEE Transactions on Computers*, vol. 55, no. 11, pp. 1410–1422, Nov 2006.

[14] D. Kim, M. Kim, K. Kim, M. Sung, and W. W. Ro, "Dynamic load balancing of parallel surf with vertical partitioning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3358–3370, Dec 2015.

[15] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic heterogeneity-aware resource provisioning in the cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 14–28, Jan 2014.

[16] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available google cluster trace," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95*, vol. 94, 2010.

[17] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 7.

[18] M. Maheswaran, S. Ali, H. Siegal, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*. IEEE, 1999, pp. 30–44.

[19] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[20] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," Technical report, Tech. Rep., 2006.

[21] Q. Kang, H. He, and H. Song, "Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm," *Journal of Systems and Software*, vol. 84, no. 6, pp. 985–992, 2011.

[22] K. M. Tarplee, R. Friese, A. A. Maciejewski, and H. J. Siegel, "Scalable linear programming based resource allocation for makespan minimization in heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 84, pp. 76–86, 2015.

[23] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, Nov 1989.

[24] T. D. Braun, H. J. Siegal, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," in *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, 1999, pp. 15–29.

[25] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh*. IEEE, 1998, pp. 79–87.

[26] U. N. Bhat, *An introduction to queueing theory: modeling and analysis in applications*. Birkhäuser, 2015.

[27] A. Erramilli, O. Narayan, and W. Willinger, "Experimental queueing analysis with long-range dependent packet traffic," *IEEE/ACM Transactions on Networking (TON)*, vol. 4, no. 2, pp. 209–223, 1996.

[28] G. Giambene, *Queuing theory and telecommunications*. Springer, 2005.

[29] N. T. Bailey, "A continuous time treatment of a simple queue using generating functions," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 288–291, 1954.

[30] F. Guillemin and J. Boyer, "Analysis of the m/m/1 queue with processor sharing via spectral theory," *Queueing Systems*, vol. 39, no. 4, pp. 377–397, 2001.

[31] O. Brun and J.-M. Garcia, "Analytical solution of finite capacity m/d/1 queues," *Journal of Applied Probability*, pp. 1092–1098, 2000.

[32] O. Kella, "The threshold policy in the m/g/1 queue with server vacations," *Naval Research Logistics (NRL)*, vol. 36, no. 1, pp. 111–123, 1989.

[33] P. A. Kivestu, "Alternative methods of investigating the time dependent m/g/k queue," Ph.D. dissertation, Massachusetts Institute of Technology, 1976.

[34] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on networking*, vol. 1, no. 4, pp. 397–413, 1993.

[35] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On designing improved controllers for aqm routers supporting tcp flows," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2001, pp. 1726–1734.

[36] T. Bu and D. Towsley, "Fixed point approximations for tcp behavior in an aqm network," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1. ACM, 2001, pp. 216–225.

[37] D. Daley, "Certain optimality properties of the first-come first-served discipline for g/g/s queues," *Stochastic Processes and their Applications*, vol. 25, pp. 301–308, 1987.

[38] D. M. Wishart, "Queuing systems in which the discipline is âĂIJlast-come, first-servedâĂİ," *Operations Research*, vol. 8, no. 5, pp. 591–599, 1960.

[39] J. R. Jackson, "Queues with dynamic priority discipline," *Management Science*, vol. 8, no. 1, pp. 18–34, 1961.

[40] L. Flatto *et al.*, "The waiting time distribution for the random order service $m/m/1$ queue," *The Annals of Applied Probability*, vol. 7, no. 2, pp. 382–409, 1997.

[41] R. Adams, "Active queue management: a survey," *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1425–1476, 2013.

[42] A. Duda, "Transient diffusion approximation for some queuening systems." in *Proceedings of the 1983 ACM SIGMETRICS conference on Measurement and modeling of computer systems*. ACM, 1983, pp. 118–128.

[43] S. Sharma and D. Tipper, "Approximate models for the study of non-stationary queues and their applications to communication networks," in *Communications, 1993. ICC'93 Geneva. Technical Program, Conference Record, IEEE International Conference on*, vol. 1. IEEE, 1993, pp. 352–358.

[44] Y. Tipsuwan and M.-Y. Chow, "Control methodologies in networked control systems," *Control engineering practice*, vol. 11, no. 10, pp. 1099–1111, 2003.

[45] A. Pitsillides, P. Ioannou, and L. Rossides, "Congestion control for differentiated-services using non-linear control theory," in *Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on*. IEEE, 2001, pp. 726–733.

[46] Y. Fan, Z.-P. Jiang, and H. Zhang, "Network flow control under capacity constraints: A case study," *Systems & control letters*, vol. 55, no. 8, pp. 681–688, 2006.

[47] D. P. Palomar and M. Chiang, "Alternative distributed algorithms for network utility maximization: Framework and applications," *IEEE Transactions on Automatic Control*, vol. 52, no. 12, pp. 2254–2269, 2007.

[48] H. K. Khalil and J. Grizzle, *Nonlinear systems*. Prentice hall New Jersey, 1996, vol. 3.

[49] S. Boyd and A. Mutapcic, "Subgradient methods," *Lecture notes of EE364b, Stanford University, Winter Quarter*, vol. 2007, 2006.

[50] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 585–599, Feb 2016.